National Technical University of Athens
Department of Mechanical Engineering

Biomechanics and Biomedical Engineering

Development of a Remotely-Controlled Autofocus System For a Microscope:

TEAM MEMBERS:

- Athanasopoulos Athanasios
- Mpantes Fotis
- Saloufas Michalis
- Varvagiannis Efstratios

SEMESTER:

9th

SUPERVISOR:

Tzeranis Dimitrios

Contents

I. General

The objective of the current work is designing and manufacturing a subsystem to control the motion of the stage of a classic microscope with respect to the vertical (z) axis and providing a basic autofocus feature. This project will be integrated with previous works on automating the fluorescent microscope *Axioskop 50 HBO* that belongs to the biotechnology lab of the department of Mechanical Engineering, NTUA.

The vertical motion of the stage of a microscope is a key parameter in automated microscopy. As it might be already known, by adjusting the vertical positon of the stage the distance between the objective lens and the biological sample is being adjusted. This operation is used to achieve clear images with no blurriness due to out of focus phenomenon. Thick samples are very difficult to be focused accurately, especially when the stage plane is not absolutely parallel to the ground. By moving the stage in the XY plane some regions of the image can be defocused and an algorithm is needed to perform small focus corrections. This procedure requires to an auto-focus procedure which relies on our system of controlling the z motion of the stage. We managed to develop such an auto-focus procedure based on simple image processing techniques. In addition an interface that integrates all the mentioned features described above was designed.

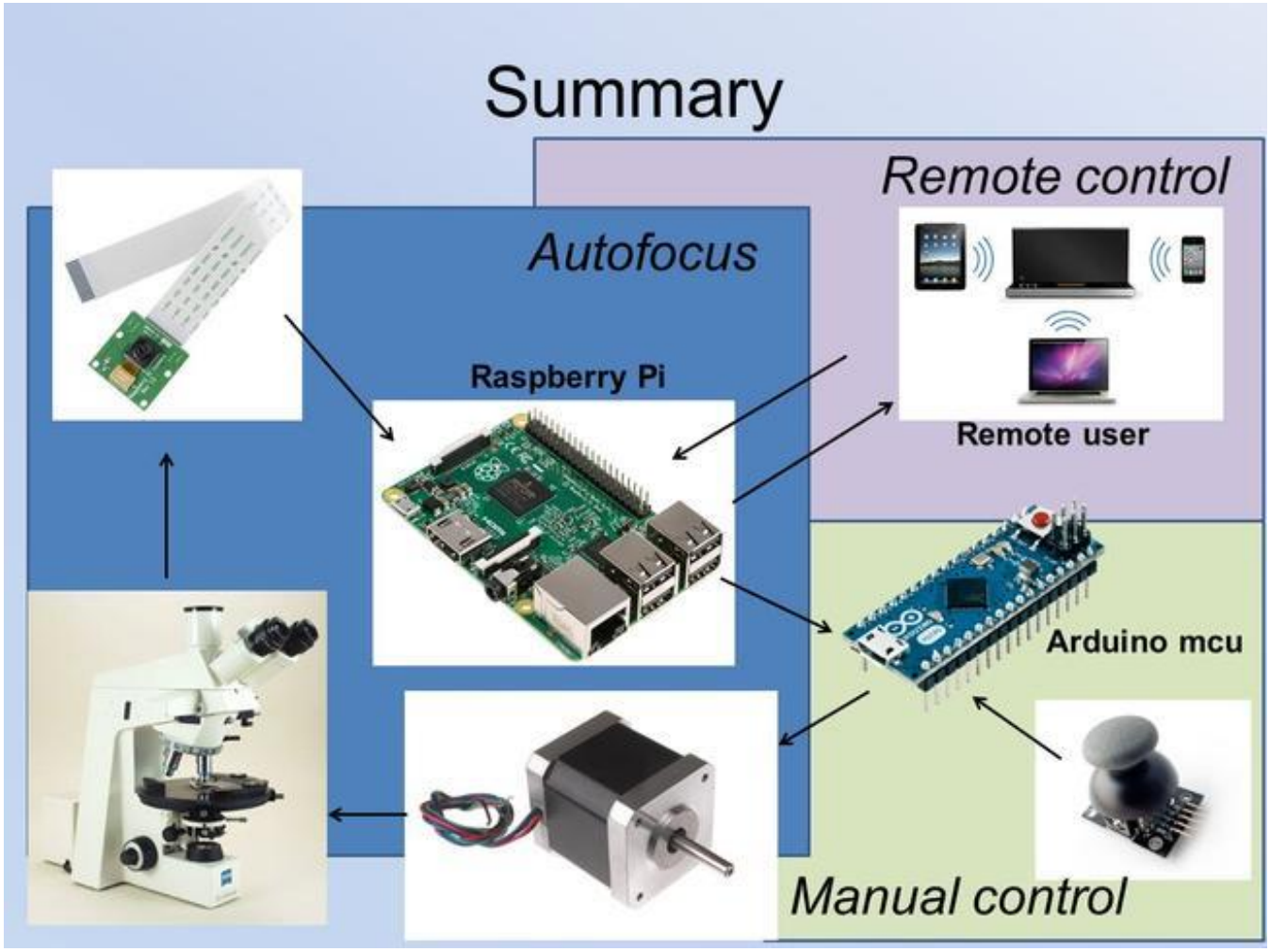
Our goals on this project are:

- A simple, sturdy and easy-manufactured mechanical system
- Low cost
- Easy to use software with no need of a dedicated PC or commercial software to control the microscope
- An open platform which can be easily extended to support relevant projects with different needs
- Remote controlling the microscope
- Remote acquisition and storing of sample images

In order to achieve the goals mentioned above we concluded to the following decisions:

- **A stepper motor** will rotate a small shaft that is used to move the stage by hand. The two shafts (motor's and microscope's) will be coupled with a commercial bronze coupler
- **A Raspberry Pi** microcomputer will be used for image processing and server capabilities
- **An Arduino Micro** microprocessor controls the stepper motor and hardware
- The **Raspberry Pi Camera** is used for image acquisition. This camera is designed especially for the Raspberry Pi and provides a lot of low level capabilities (for example controlling the shutter speed) with the use of already developed modules.
- The Raspberry Pi will act as a local HTTP server running the open source **Apache 2.0 web server**
- Image processing algorithms will be written using the open source vision library **OpenCV 2.0** available for C++ and Python developers
- The software for image processing and serial communication between Raspberry and Arduino will be written in **Python** as a lot of modules are already developed

- The interface will be developed in **HTML-PhP** and will be available to any wireless device connected to the lab's network via the Apache server. On the client side (wireless device) only a compatible (Safari, Mozilla) browser is needed.



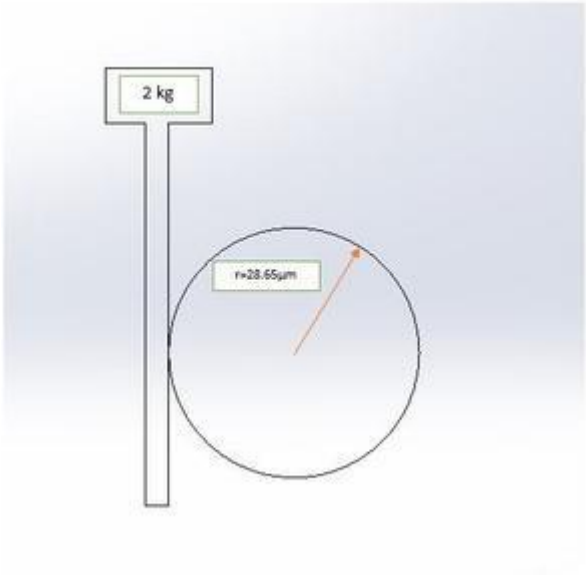
Limitations:

- HTTP protocol doesn't provide real time data exchange capabilities. In our project such a capability is not necessary needed, but may be needed for remote closed loop control of the stage position or wirelessly transferring live image processing data. For such features WebSockets or general socket based on TCP/IP protocol must be considered.
- No closed loop control of the stepper by using an encoder i.e. Assumptions that the stepper will not lose steps were made. The reason is fully described on later chapter.
- As the image is stored on Raspberry Pi's local disk (SD card) and not in RAM there are speed limitations. Although image storing is not needed at very high frame rates.

II. Hardware Part

Microscope Stage Motion System Modelling

The vertical motion of the stage is adjusted by rotating a small axle, which means that a transmission system interferes. The transmission ratio is i=180μm/rev and the stage's mass is estimated at 2kg. Assuming that the inertia of the transmission system is negligible, the following simplified model will be used:



The equivalent distance(as shown above) of the stage from the axis of the rotating shaft (or the gear's radius) is calculated as follows:
1 rev -> 180 μm thus (2π)R = 180 μm --> R = 28.65 μm.

The equation that describes the systems behavior is: $(J+mr^2)a + mgr + T_f = Tin$

where :

- a is the angular acceleration (rad/s²)
- J is the total inertia of the motor's shaft (J_R) plus the inertia of the microscope's shaft Js, so J=Js+Jg

- the inertial term mr² is proportional to 10^{-10} kgm² so is negligent
- mgr=19.62*28.65x10-6=0.56x10^{-3} Nm

T_f is the friction loses of the system. We are not able to calculate the exact value, but it is anticipated that they are quite small, considering that they depend on speed which will be low. A generic model like $T_f = (B_R + B_L r)\omega$ where B_R, B_L are the loses factors for the rotational and the linear motion can be applied. The exact value of these coefficients can be estimated experimentally but we can assume a value of B=0.0005 for the overall (linear and rotational) friction coefficient.

Calculation of Js

The shaft consists of a cylinder with diameter equal to 3mm and length 12mm and a second one with diameter equal to 8mm and length 22 mm (see sketch below). Assuming that the shaft's material is steel (density = 7800 kg/m²) we can calculate the mass and therefore the inertia of the two cylinders.

- m_1= 0.012x7800(πr_1^2) = 0.66x10^{-3} kg
- Js_1= 0.5$m_1 r_1^2$=7.4x10^{-10} kgm^2

- m_2= 0.022x7800(πr_2^2) = 8.6x10^{-3} kg
- Js_2= 0.5$m_2 r_2^2$=688x10^{-10} kgm²

Thus:

Js =695.4x10-10 kgm2 = 0,07 10^{-6} kgm^2

It is important to note that the inertia calculated is significantly smaller than the rotors inertia which is found from the motor's specifications as J_R=5.4x10^{-6} kgm^2

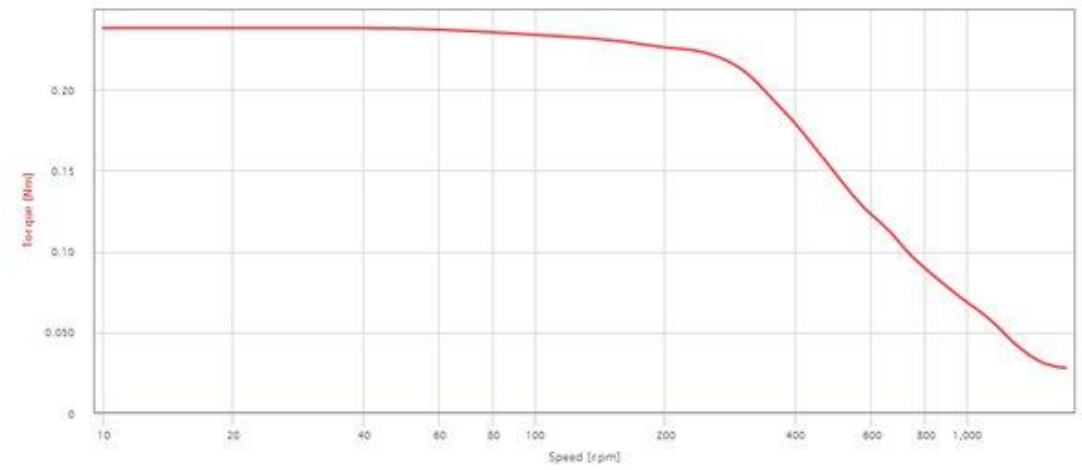
The overall inertia for the motor-microscope coupled system is approximately J = 5.5x10^{-6} kgm²

Stepper Motor Sizing

In order to define if the specific stepper motor is appropriate for our task, the following specifications should be taken into consideration:

- The rotation of the shaft is very slow (approaching the speed that it is rotated by a human hand). n=10-20 rpm
- The stepper motor shouldn't lose any steps.
- We should be able to reverse the speed vector quickly.

The torque-speed curve of the motor is shown below(for 1.2A, 24V) :



***Max torque**:* 0.238 Nm

The torque needed to accelerate our system it is definitely below the red curve so the stepper motor would not lose synchronism during acceleration due to system's inertia.

Our motor will operate in low speeds (10-20 rpm).This range is called **locked step mode,** which means that it decelerates between each step having to ability to reverse its movement instantly. Operating into this range, the stepper is not losing any steps.

For a basic inertial calculation we assume a max speed of 20 rpm = 2,09 rad/s. Our stepper motor has a step of 0.9° so we will be sending:

f = 2.09 rad * (180/π) / 0.9 = 133.05 Hz or pulses per second.

Assuming 1/3-1/3-1/3 accelerating-steady state-decelerating profile for each step the acceleration is calculated as:

a = 2.09 rad/s / (0.333*(1/133.05)s) = 834 rad/s².

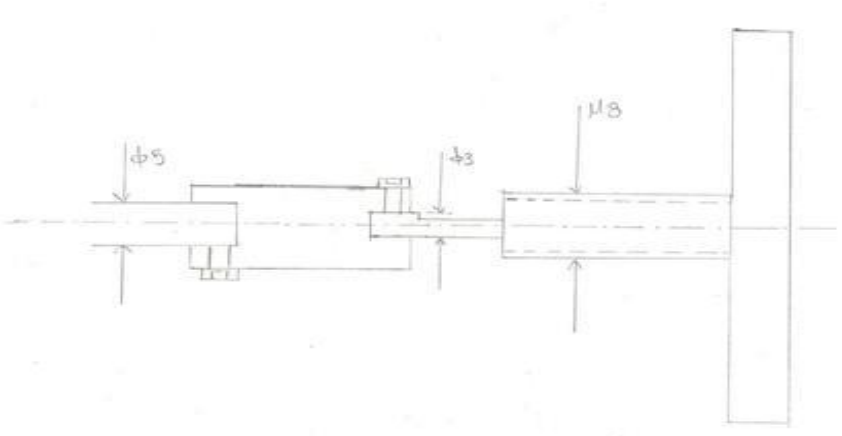
The needed torque using the inertia calculated above for this acceleration is :
T_{in}=4.5x10^{-3} Nm much smaller than the max holding torque of the motor so that the motor can accelerate and decelerate the load in the locked step region (the moment due to the mass of the stage is 10 times smaller than the moment required to accelerate the stepper so is not calculated). So the inertial phenomena are negligent.

During the steady state motion we have to overcome the moments due to the weight of the stage and the friction loses. As calculated above, this moment is about:
T = mgr + Bω = 1.5 10^{-3} Nm

Connecting the two shafts

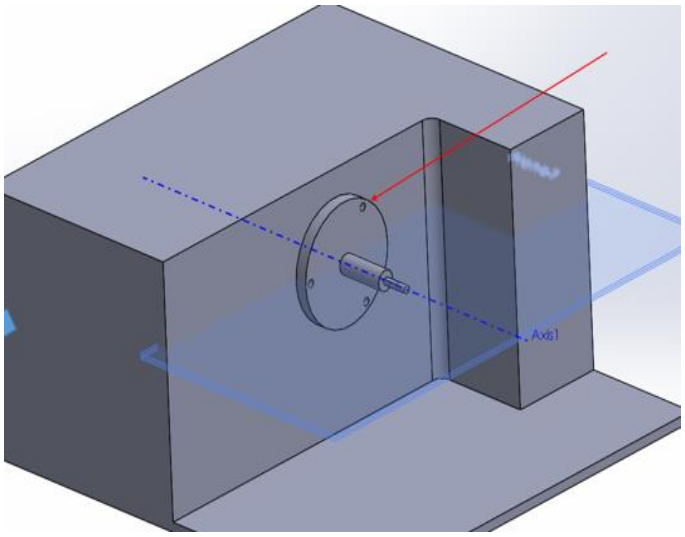
In order to connect the rotor shaft with the microscope's we will need a coupler (there is no need for an elastic coupler) with a 3mm hole and a 5mm hole. Because such a coupler was not commercially available, one with 3mm hole on both sides was modified to fit our situation.

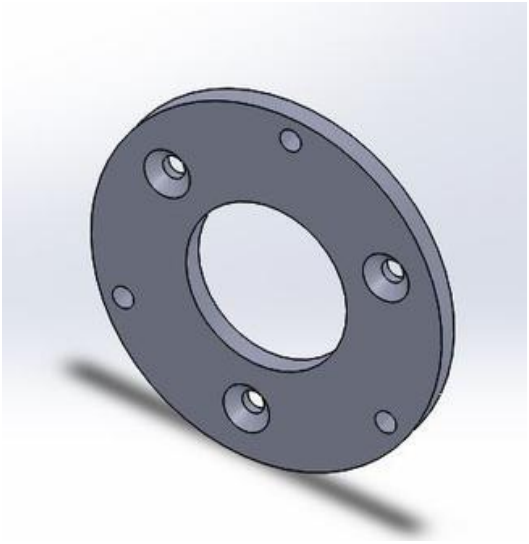
A sketch of the coupling of the two shafts is the following:



Motor attachment

The design of a small structure is needed in order to couple the shafts of the stepper motor and the microscope as described in the section above. On the bearing on which the microscope's shaft is based there are three holes with M3 thread as shown next which can be used to fasten our structure to the microscope's body and achieve linearity between the two shafts



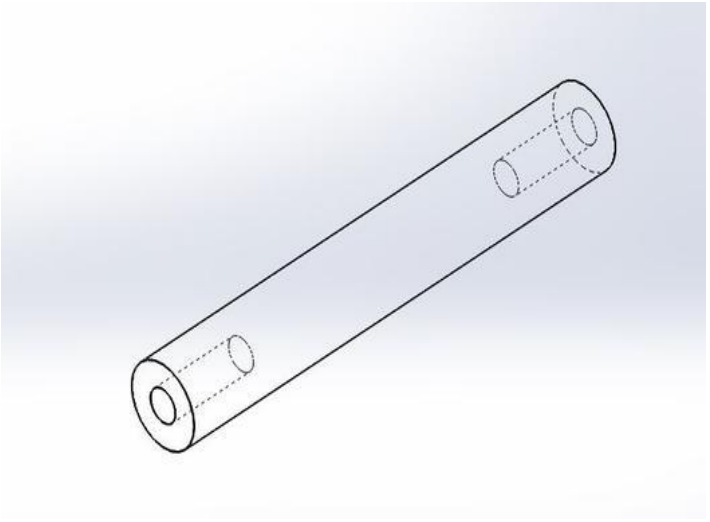


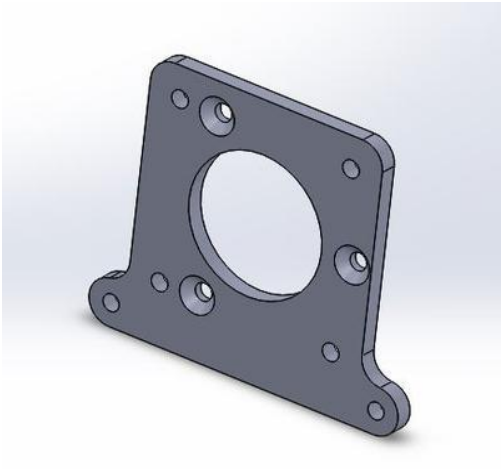
This flange is mated with the microscopes body, taking advantage of the three M3 holes. There are two groups of three holes, with a conical section in order to fit countersink bolts.

Three bolts will be used to hold our structure on the microscope’s body, and the rest will be fastened to the following part.

Three identical parts were manufactured in a lathe using aluminum as material to ensure the strength of this structure and achieve small displacements.

This design is chosen due to its low cost, fast and easy construction.

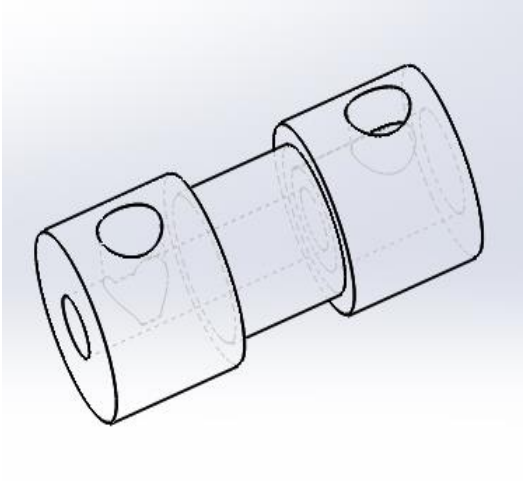




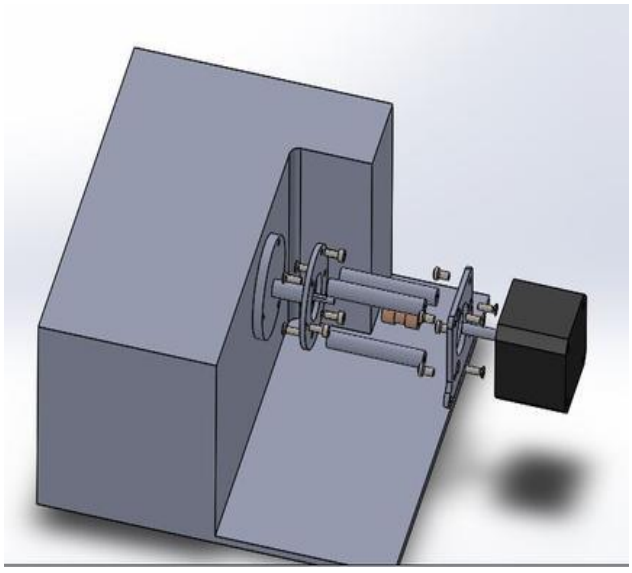
In order to mate the motor with these cylindrical parts another flange is designed that matches the geometry of the motor's front view (see motor specifications for a detailed drawing of the motor front face - technical documentation [2]).

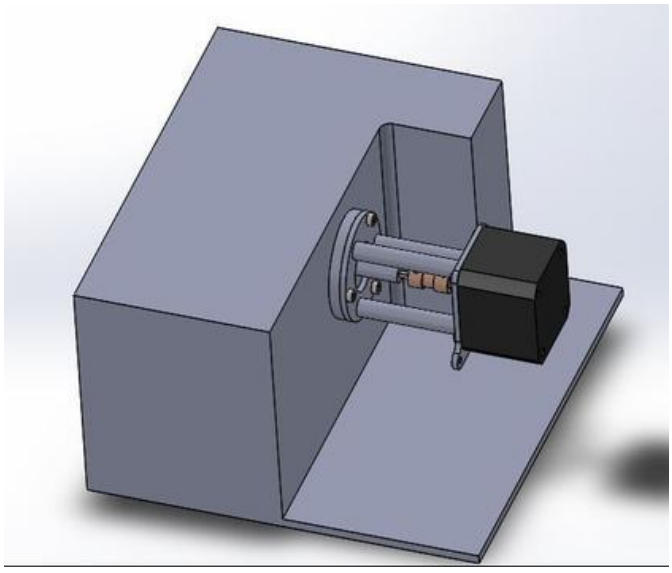
The above part is fastened on the motor with 4 M3 allen-type bolts, as well as with the 3 cylinders using M3 countersunk screwdrived bolts normal to the longitudinal direction of each of these parts.

Finally, in order to couple the motor's shaft with the microscope's shaft a commercial 3mm to 3mm bronze coupler was used. Our task required a 3mm to 5mm coupler, so it was modified appropriately by converting one side of the couple to 5 mm, using a simple drilling process.



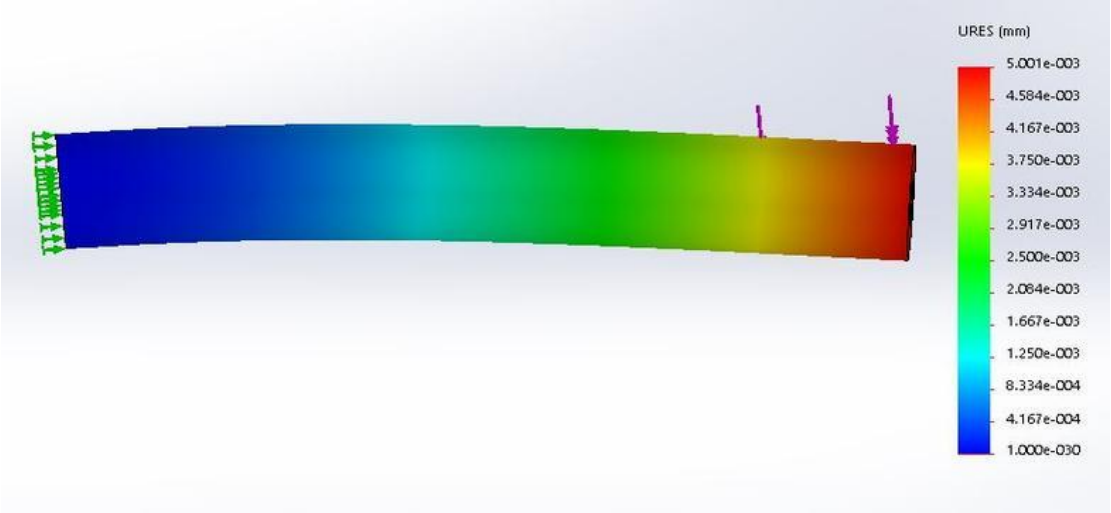
The mating of the parts that was previously described, is shown below:





Static Analysis

A static analysis is performed (using a simple FEM analysis in Solidworks) in order to determine the sturdiness and the bending deflection of this structure. Large bending deflections can cause loss of linearity between the two shafts.
The maximum load for each cylindrical part is estimated at 1.5 N and the material aluminum.

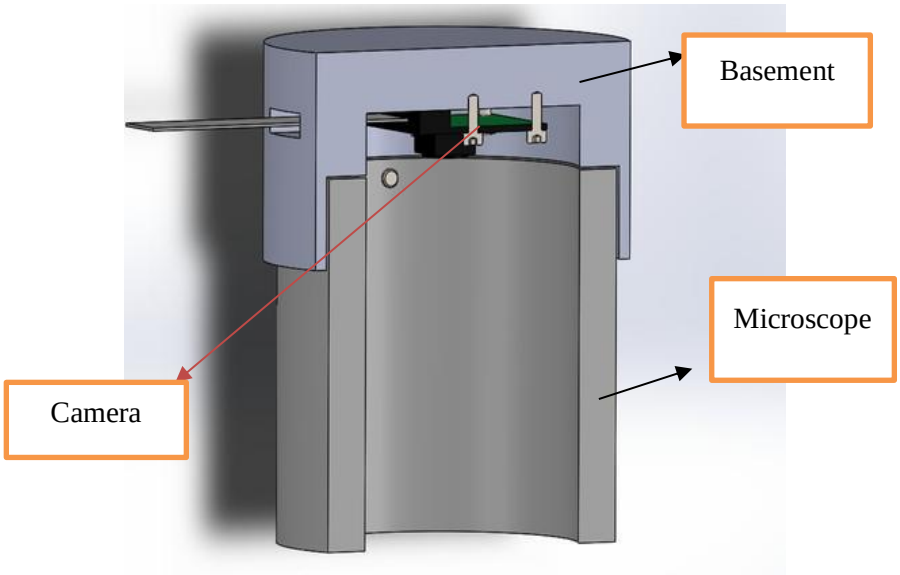


Results:
The above figure shows that a deflection around 5 μm is calculated at the end of the part, which is acceptable regarding the manufacturing precision of each part.

The angular deflection of the cylindrical parts is calculated at 0.005^{o}, so the two shafts can be considered linear.

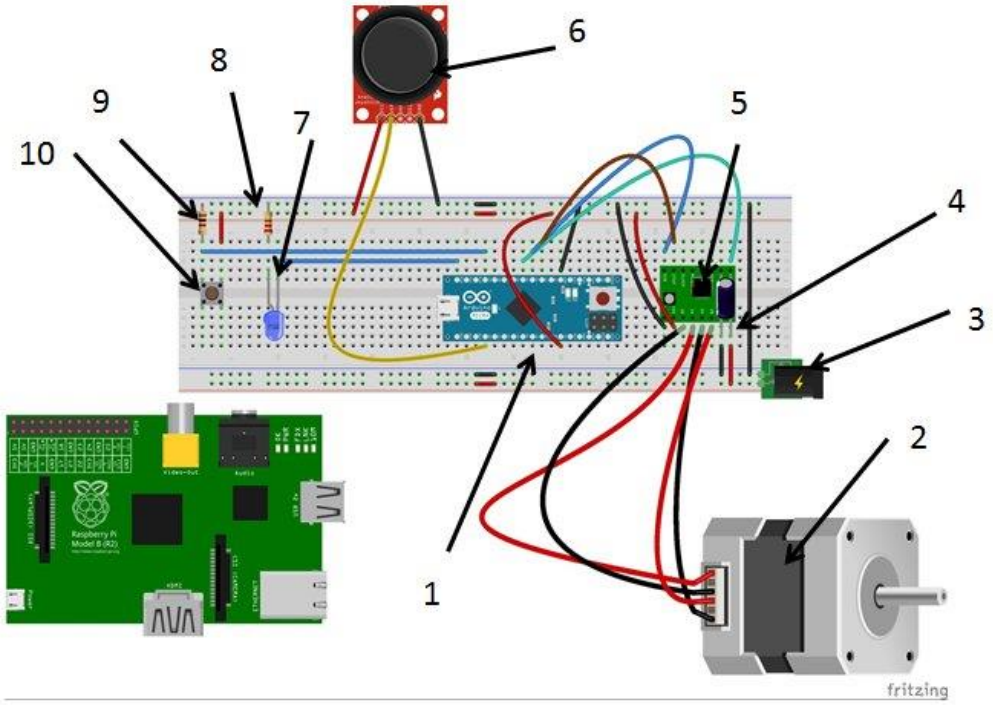
Pi Camera Basement

In order to mount the camera on the microscope a 3D printable basement was designed. A 3D drawing of this basement follows:



Electronic Circuit and Wiring

A figure of the circuit we used follows:



The description and usage of each component is described in the following list:

1. **Arduino Micro**
2. **Stepper motor:** Wired in bipolar according to the manufacturer's drawings (see appendix)
3. **Jack for external power supply:** It is used in order to provide the necessary power for the motor. An **24 VDC- 1200 mA/ center minus** power supply is recommended
4. **100 μF capacitor:** Used to smooth voltage instabilities of the power source
5. **Stepper motor driver:** Is a cheap A4998 driver, with maximum current per coil equal to 2A
6. **Joystick connected to analog input of Arduino:** Taken from a PS3 replica controller
7. **Indicating LED:** Used to indicate the that the autofocus operates
8. **220 Ω resistor:** To avoid the destruction of the LED due to overcurrent
9. **1 kΩ resistor:** Protects the switch from overcurrent
10. **Switch for initialization of the Z=0 position**

A more specific electronic drawing is appended on the Appendix.

1. Comments on the bipolar wiring of the motor

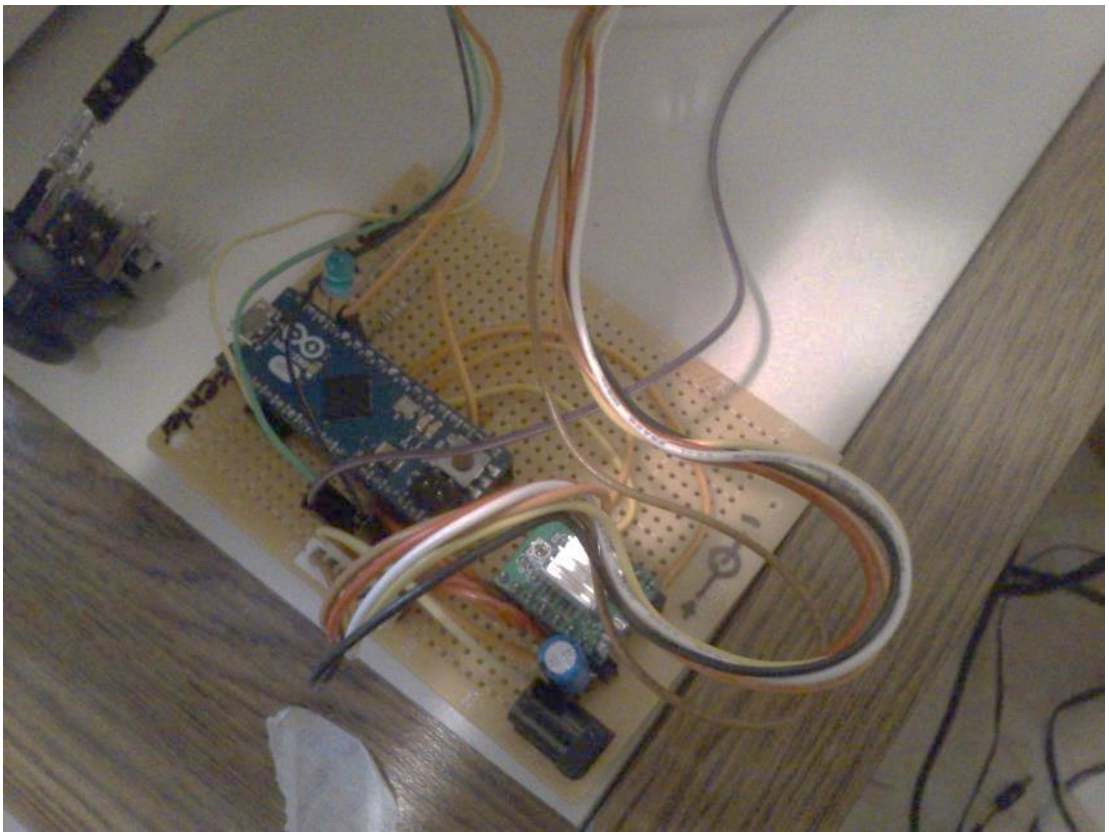
There are two ways to drive the stepper motor, either unipolar or bipolar. Bipolar connection is preferred due to the fact that more torque can be produced this way, especially in low speed. By wiring the motor in bipolar, the current flows through the whole coil which results into a stronger magnetic field and therefore to higher torque. On the other hand, unipolar connection takes advantage of only half of the coil which is quite simpler, but not as effective as bipolar.

2. Comments about the joystick

A classic joystick like the one that was used consists of two potentiometers whose resistance varies according to the joystick's XY position. As we are interested only for one dimension motion we wire only the one potentiometer as it was described above. The middle pin of the potentiometer is used to determine the position by voltage division (that's why it is connected to an analog input of the Arduino board). The other two pins are used for the voltage (5V) and the ground.

All the electronic components were soldered on a board, as it is shown in the next figure.

Note that two different voltages should be measured on the board: 24V across the pins of the power supply and 5 V across the Arduino VCC and GND pins. The grounds must be common. The way the power supply jack was soldered a **center minus power supply** must be used **(important!)**.



III. Software Part

Image Acquisition Script (ImageAcqusition.py).

At a fixed time rate the Pi Camera takes images of the samples using the OpenCV functions for image acquisition. This rate is defined with the constant FPS in the ImageAcqusition.py python script and is set at the value of 10 fps, which found that is satisfying by means of sampling and wireless transmitting. The image is being saved on Raspberry Pi's local disk (SD card) both in compressed (jpeg) and uncompressed (bmp) format. The bmp format is used for image storing and processing at later time and the jpeg format is used for live wireless transmission.

The python script that is used for the above described procedure is appended on the appendix (ImageAcqusition.py).

A variety of solutions is available in order to make a video stream available on the local network. Possibly a pretty simple solution which does not require a lot of low-level coding and will moreover work well both for web browsers or self-programmed client applications for receiving the stream is using the Mjpeg technology, which in simple words consists in streaming a sequence of simple jpeg images with no need for video compression (assuming that we are not interested about transmitting sound). A simple implementation of this technique is to continuously take a picture with the camera and store it to the same file each time using the jpeg encoding. A script watches when this file changes and since this occurs transmits the file via an http file server.

Thankfully there are some implementations of this technique available as open source applications, just like mjpeg-streamer. Mjpeg-streamer provides both an event viewer (checking when the image file is changes) and an in-built web server that streams over the port 8080 of the Raspberry Pi. The general syntax for mjpeg-stramer application for our case is:

```
LD_LIBRARY_PATH=/usr/local/lib mjpg_streamer -i "input_file.so -f [directory containing the changing image] -n [changing image filename].jpg" -o "output_http.so -w /usr/local/www"
```

As we use the Apache web server for wireless communication we need to reverse proxy the mjpeg-streamer port to Apache's port (by default 80). This can be done with the **mod_proxy module** of Apache web server. In other case the image stream will not be available on the web interface, but only in the port 8080. More details follow in next Chapter (*Web Interface and Apache Server*)

Serial Communication Script (Com.py)

This script is quite simple as raspberry acts just like a simple echo server. The script is waiting for a standard command pattern of the form:

python [scriptname] -X[num1] -Y[num2] -Z[num3] -A[-1 or 0 or 1]

and simply echoes over the serial port that reads from the file **serialPort.dat** (see testCom.py script) each argument and it's specific value. For example if we run the following command

python Com.py –Z100

the following will be transmitted over the serial bus:

Z100

The Arduino acts as a slave device waiting from Raspberry Pi the appropriate command which has a special meaning according to the Arduino program. For example the above example means that the Arduino sends to the stepper which controls the Z motion of the stage the command to perform 100 steps clockwise. As it is obvious the effect of each command is relevant to the Arduino code which is described in next chapter.

The Com.py script is appended to the appendix.

Verifying the correct serial port (testCom.py)

Often the serial port on which the Arduino is connected changes, especially when the Arduino IDE runs or multiple Arduino boards are connected to the Raspberry. In order to find the correct Arduino board which contains the software we developed, this script was written.

The testCom.py script lists all the serial devices of the Raspberry Pi and tries to connect by sending a predefined message (the character 'T') to each of them and waiting for 2 seconds. If the device replies with a predefined message (actually this message is simply an 'OK') this is the Arduino device we are going to use. The detected serial port is saved on the file **serialPort.dat** of the working directory.

Image processing and auto focus script (autofocus.py)

This ***script integrates the previous scripts of image acquisition and serial communication with Arduino with a simple image processing*** **algorithm** creating an auto focus procedure. The image that is stored in Raspberry's disk by the ImageAcquisition.py script is read by this script. Then an image processing procedure is performed in order to determine if the image is blurred or not (more details on the next paragraph). The motion of the stage between different positions where images are stored relies on the Com.py script which sends commands via the serial bus to the Arduino program to control the stepper.

Many different ***image processing*** algorithms are described in the literature either simple or more sophisticated. In our case a very basic algorithm to detect the blurriness in the whole image plane and retrieve a single value to determine if an image is blurred or not was preferred for its simplicity.

This algorithm consists on ***the variance of the Laplacian*** [2] of the whole image. The Laplacian image is the image that occurs after applying the Laplace operator on each point of a grayscale image, or convolving the image with a Laplace kernel like the following:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

It is already known from basic image processing that the Laplace operator is used to detect the edges in an image. An example of a picture and its Laplacian follows verifying this statement.

The next step is to calculate the variance of the Laplacian image by using the following expression:

$$LAP\_VAR(I) = \sum_{m}^{M} \sum_{n}^{N} \left[|L(m,n)| - \overline{L} \right]^2$$

where $\overline{L}$ is the mean of absolute values given by

$$\overline{L} = \frac{1}{NM} \sum_{m}^{M} \sum_{n}^{N} |L(m,n)|$$

Thankfully this procedure takes only a line of code on OpenCV!

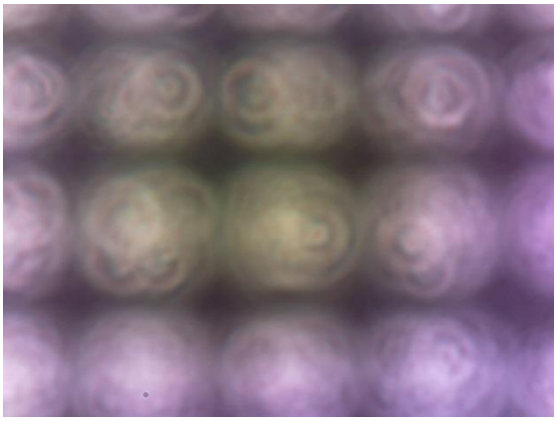
```
bv = cv2.Laplacian(gray, cv2.CV_64F).var()
```

So in an image with sharp edges (which is a more focused image) a greater variance value is calculated than in pictures more defocused. In other words by using this algorithm our goal is to find the Z position where:

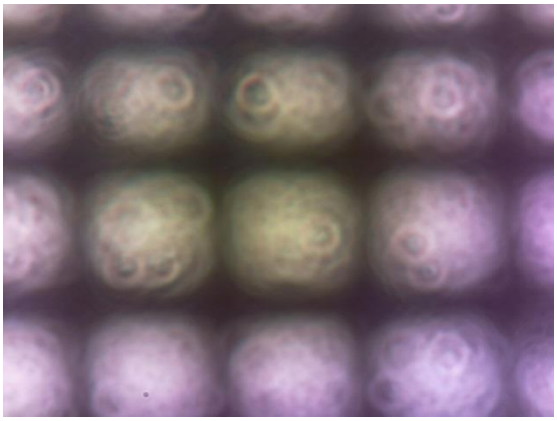
$$F = LAP - VAR = MAX$$

which implies to a classic optimization problem. Ideas on how to find this position are proposed to different papers where auto focus techniques are discussed. In our case a simple procedure of starting of a point and moving until the next taken image is more blurred (so its LAP_VAR is lower than the previous' image) and then the stage will return to the previous position. This method is not very robust but it is simple and worked for this project.

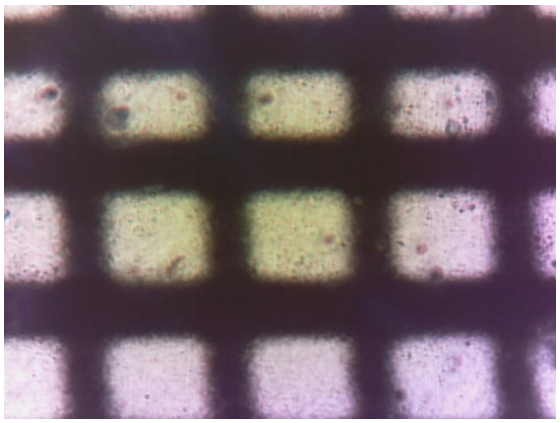
To be more specific an example follows:



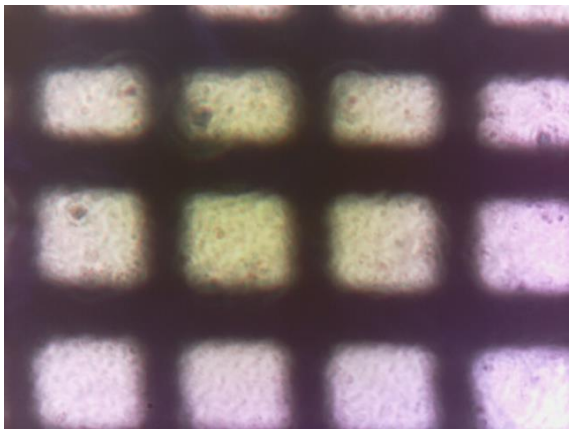
1. LAP_VAR = 32.032



2. LAP_VAR = 33.780



3. LAP_VAR = 40.698 = MAX
➔ (On focus)



4. LAP_VAR = 36.863

Finding that in the 4th position the image has a smaller value of LAP_VAR than its previous' (meaning that it is defocused) the algorithm stops and returns the stage to the previous position.

The number of steps between two discrete positions (i.e. position 2 and 3) of the stage can be adjusted with the constant N_STEPS of this script. The default value in our case is 200 steps or 180°, in order to test robustly the algorithm. When the in-focus position is found

the algorithm can be run again with a refined N_STEPS (smaller than previous), so a more accurate in focus position can be found.

The Arduino Program

In the previous sections we described the software on the Raspberry Pi's side. On the Arduino side the program written performs the following:

- **Serial Communication** with Raspberry Pi
- **Control of the stepper motor**, either by **serial communication** or by reading the connected **joystick**
- **Checks if a number of steps is exceeded** to avoid possible destruction of the lenses by the stage
- Controls the indicating **LEDs**

The first part of the program (in the loop function) checks whether there are available bytes in the serial buffer. If this is true then reads the first character.

In case that this character is equal to 'Z' then it expects a following integer which represents the number of steps to be performed. If this number is positive then the shaft rotates clockwise else it rotates counterclockwise. In order to control the stepper motor the external library *AccelStepper.h* was used. The motor rotates at a constant speed of 20 RPM which is represented by the global constant **RPM** on the Arduino code. 'X' and 'Y' arguments doesn't be coded yet as the above program is used solely to control the Z stepper but can be extended easily.

In case that the incoming character is 'A' then the program is waiting for an integer. Three cases are possible:

1. 1 means that the autofocus script is running. The autofocus procedure starts from the position Z=0 (we xill explain later how this position is defined) and the autofocus indicating LED is turned on
2. -1 means the autofocus script is running but in this case the autofocus procedure starts from the current position. The autofocus indicating LED is turned on
3. 0 means that the autofocus procedure has terminated and the picture that is taken is focused. The autofocus indicating LED is turned off.

Moreover the motor can be controlled manually via the joystick. A function called *joystickWatcher()* checks if the reading value from the analog input differs from zero and maps this value to the interval [-5,5]. In this way we have 5 discrete speeds to run the motor on each direction, by changing the delay time between each loop.

In both cases a check is performed that no more than **maxAllowedPos** steps are being performed from the Z=0 position. As mentioned above this check is performed for safety reasons about the lenses. If the **maxAllowedPos** steps are exceeded then the indicating LED blinks and the shaft is not rotating. By default this constant has the value 2400 (so as the stepper motor has an angular step of 0.9° or one full rotation is 400 steps we have 6 full rotations available. This number is small and in fact has this value only for debugging purposes and must be changed according to the user's preferences.

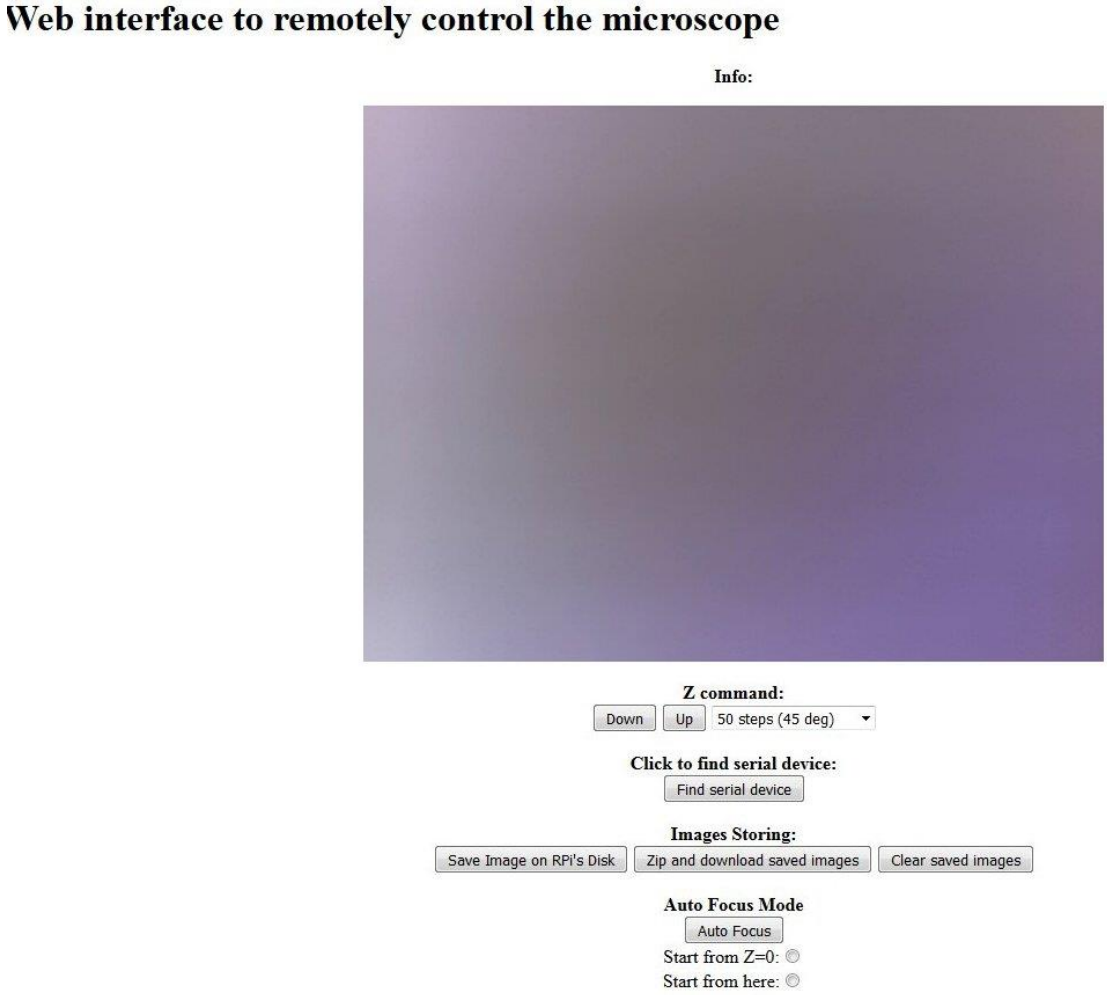
But how we define the Z=0 position and we know the number of steps being performed from this position? Simply we install a push button beyond the stage. On the first run (function setup()) the motor rotates clockwise (or the stage is moving downwards) until this button is pressed. Then the stage stops and the position Z=0 is initialized. Every time this button is pressed the rotation stops as we reached the 0 position.

Web Interface and Apache Web Server

As mentioned in the general section we decided all the above functions to be enabled wirelessly, so anyone can control the microscope remotely from a wireless device connected on the lab's network and not needing to connect a monitor and keyboard to the Raspberry Pi. Moreover the user can see a live stream of the sampling images and store and afterwards download them on his device as a compressed (zip) file.

In order to provide wireless capabilities to our system Raspberry Pi must run as a server over the local network. The simplest and more robust way to do this is by running the Apache HTTP server on the Raspberry Pi. After installation, the Apache server runs each time the Raspberry Pi boots providing the files under the directory /var/www/html to the network.

An HTML form will integrate all the above mentioned software features providing an easy to use interface that will be accessible to any kind of wireless device connected to the lab's network. The design of this form is figured in the next picture:



This form is coded in PHP, running on the Raspberry Pi by installing the ***mod_php*** module of the Apache server. The PHP code runs the above described python scripts with the command:

```
exec ("python [script_name].py –Arg1 –Arg2 …");
```

Each time a button is clicked the form is submitted to the server and the appropriate action is taken. Error information is shown above the whole interface.

As we see this interface also includes the live stream of the sampling images. This works only when we reverse proxying the mjpeg-streamer's 8080 port to the Apache's 80 port with the ***mod_proxy*** module of the Apache server, as mentioned above. Unfortunately the mjpeg technology is only supported by the Mozilla Firefox and Safari by the time, so these browsers are required to be able to see the image stream.

In order to reverse proxy the port 8080 to the port 80 using the mod_proxy module add the following lines to the /etc/apache2/sites-enabled/000-default.conf file

```
ProxyPass /Rpi http://0.0.0.0:8080/
ProxyPassReverse /Rpi http://0.0.0.0:8080/
```

For simplicity a 000-default.conf file with these lines is given in the installation .zip file which should be copied to the above mentioned path.

As we have already described the user can store and download a number of sampling images by the following procedure:

- By clicking the button "***Save Images on RPi's disk"*** the user stores some images on the Raspberry Pi's SD card. This is done by running the Unix system command ***cp*** form the PHP code and copying the current frame.jpg and frame.bmp images to the current working directory (/var/www/html) of Apache server. The filenames of the copied image files are given by the current *day and time* in the following format

  d-m-y_H:i:s where

  - d stands for day
  - m stands for month etc...
- After saving a number of images the user can create a .zip file containing them and download it on his own disk by clicking the ***"Zip and download saved images"*** button. To zip the files, the PHP program runs the Unix ***zip*** command which should be installed on our Raspberry
- Finally the user is recommended to clear the stored and downloaded images in order to free space on the Raspberry's SD card and moreover avoid downloading again the same files by clicking the button ***"Clear Saved Images"***

Installing the software

In order to run this project some software must be downloaded and installed on the raspberry Pi 2. To be more specific the installation packages that are required are:

- OpenCV 3.X or OpenCV 2.4 and above (instructions can be found on links of this' project's website)
- Download the source code and install mjpeg-streamer (instructions on the website following the link)
- Apache Web Server 2

  sudo apt-get install apache2
- The required apache modules

  sudo apt-get install php5 libapache2-mod-php5
  sudo apt-get install libapache2-mod-proxy-html
  a2enmod proxy (enables the proxy module)
- A python interpreter (usually is already installed)

  sudo apt-get install python
- The picamera python module

  sudo apt-get install python-pip (if have not already installed pip)
  sudo pip install "picamera[array]"
- The zip program

  sudo apt-get install zip

When the required software listed above is downloaded and installed then download the ***Software.zip*** file ***from the website of this project*** and unzip it. The directory structure of this file is resembling the Rapsbian system directory structure, so copy each file to the corresponding directory of your system, or alternatively unzip the file into the / directory replacing any already existing files and creating any new directories that doesn't exist.

The source files are under the directory /var/www/html which is Apache's default working directory. In the /var/www/stram directory the currently acquired image (frame.jpg and frame.bmp) is stored .All the files in the /var/www/html and /var/www/stream directories must belong to the apache user, so run the following command for both these directories

sudo chown –R www-data:www-data /var/www/{dirname}

The mjpeg-streamer and the image acquisition script are running from the startup of the Raspberry. This can be done by editing the /etc/rc.local file (the appropriate file for this project is included in the Software.zip file).

Important: If you have other applications running at startup by this way please do not replace this file but only copy and paste the appropriate lines that start the mjpg-streamer and image-acquisition script to yours rc.local file.

Note: In order to run applications on startup this way you must start the raspberry on text mode. If this is not set by default please change it using *raspiconfig*.

Finally in order to allow Apache to run the python scripts for serial communication, www-data must be added to the dialout user group with the following command

sudo usermod -a -G dialout www-data

Using the software

After the software installation you only have to start Raspberry (if you have turned it off) and open a browser window from a remote machine (by the time Mozilla and Safari browsers are supported for the image stream). On the address bar of your browser insert the Raspberry Pi's ip address. You must see the user interface that was described previously.

If you don't know the Raspberry Pi's ip address then you can find it by running the following command:

```
ifconfig wlan0
```

This requires at least a monitor connected over the pi with HDMI and a keyboard. A different way is to connect the pi by LAN on a machine (the machine must act like a server by giving ip address to the Pi) and connect to the Pi via secure shell (ssh). Then you can type the above mentioned command to determine the ip address of the Raspberry Pi on the wireless network. More information about this can be found on books like the *Raspberry Pi Cookbook (Simon Monk).*

As the ip address of the Raspberry Pi may change on a DHCP (dynamic ip address) network it may be hard to know the ip address of the Pi each time. A solution is to give a static ip address to the Pi. Usually the router will give the same ip address to the Raspberry Pi though.

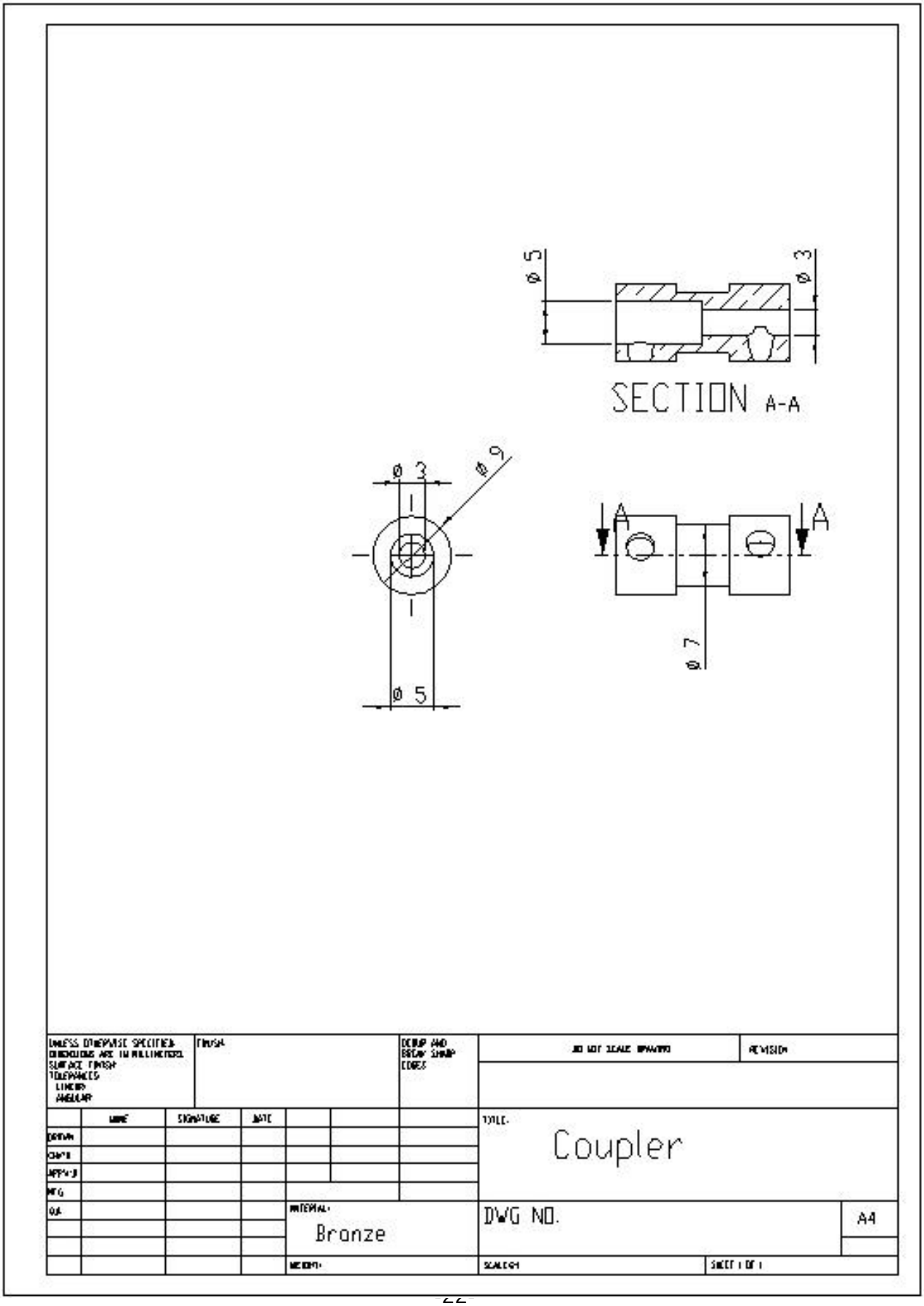
What is needed to run the project

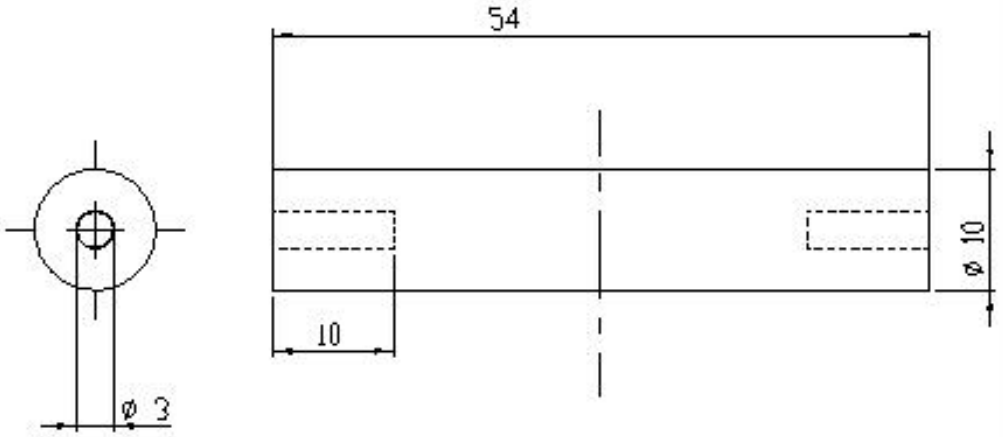
Except the Arduino micro and the Raspberry Pi boards, the stepper motor and the components we manufactured (soldered electronic board, motor mount construction) the following peripherals are needed to run the project:

1. A 18V-24V DC, 1200 mA (or greater) center minus **power supply** in order to run the stepper motor
2. A 5 V DC, 1000 mA (or greater) micro USB **power supply** for Raspberry Pi
3. An at least 8GB **micro SD** card which is used as the Raspberry's hards disk
4. A **USB Wi-Fi dongle** for Raspberry Pi
5. An **FTP Ethernet cable** to connect the Raspberry Pi to a PC via Ethernet
6. A **micro USB to USB cable** to connect the Arduino to the Raspberry

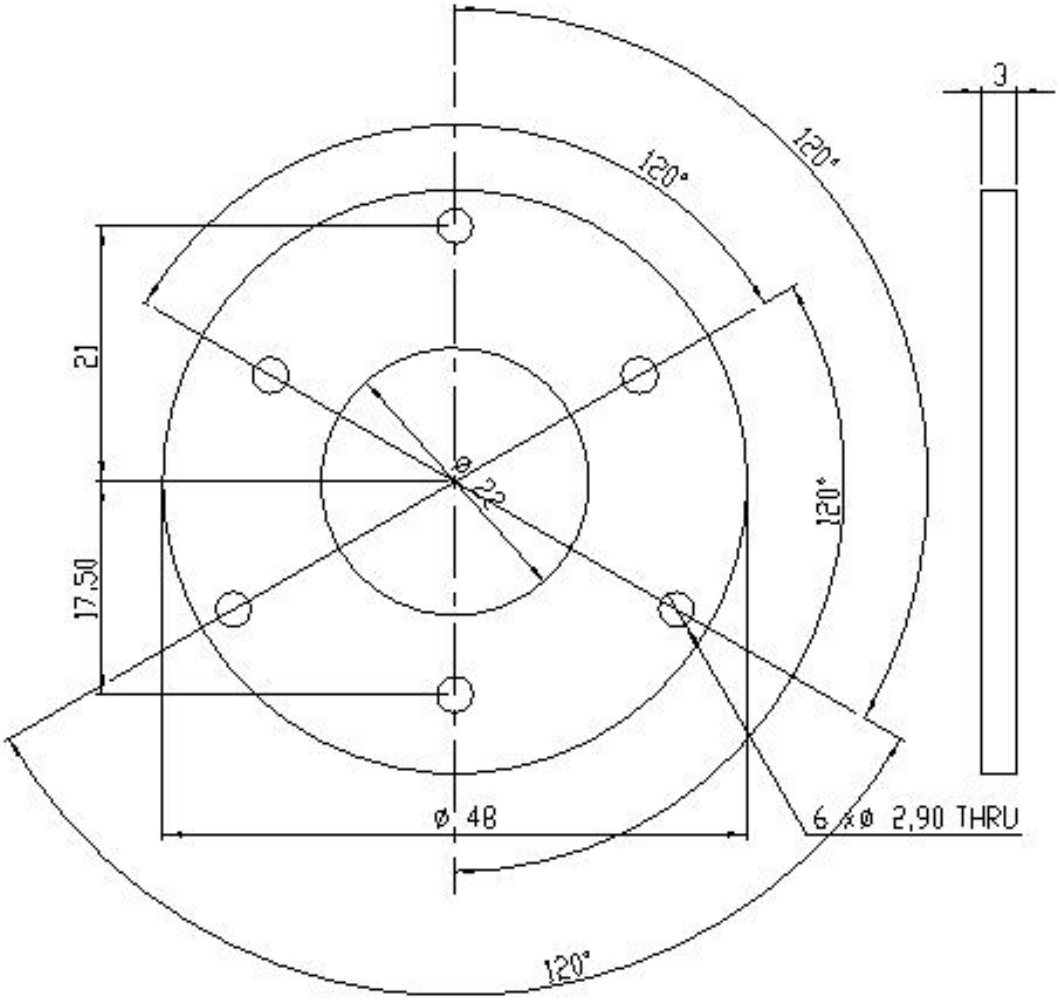
IV. Appendix

1. Drawings

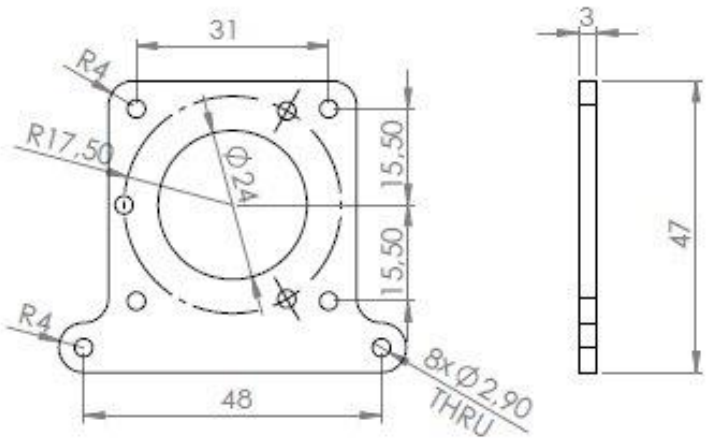




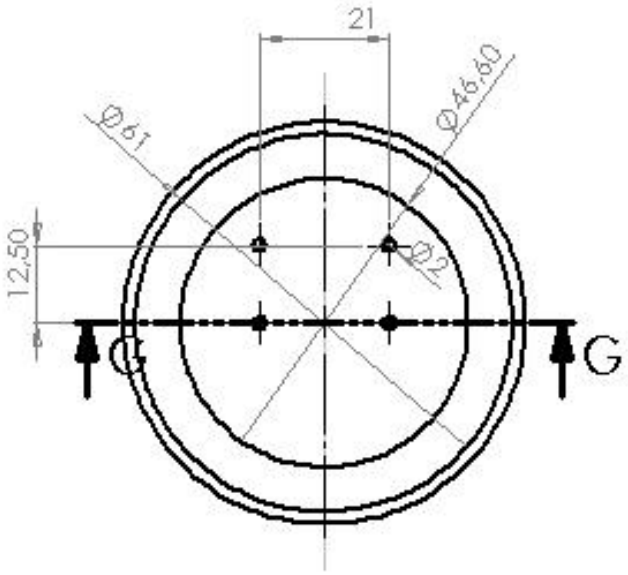
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH TOLERANCES: LINEAR ANGULAR	FINISH			DEBUR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE			TITLE: Cylinder	
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A.				MATERIAL: Aluminum		DWG NO.	A4
				WEIGHT:		SCALE:1/1	SHEET 1 OF 1

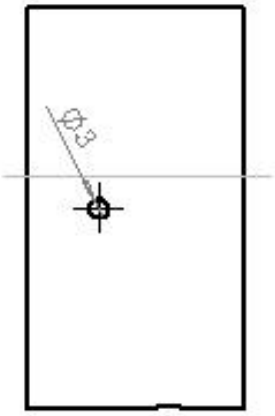


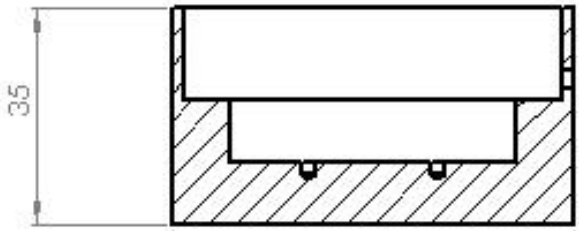
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:			DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE		TITLE: MIcroscope flange	
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A.				MATERIAL: Stainless Steel	DWG NO	A4
				WEIGHT:	SCALE:2:1	SHEET 1 OF 1



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:				DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE				TITLE:	
DRAWN								
CHK'D								
APPV'D								
MFG								
Q.A				MATERIAL:	Stainless Steel		DWG NO. MotorMount_DT	A4
				WEIGHT:			SCALE:1:1	SHEET 1 OF 1



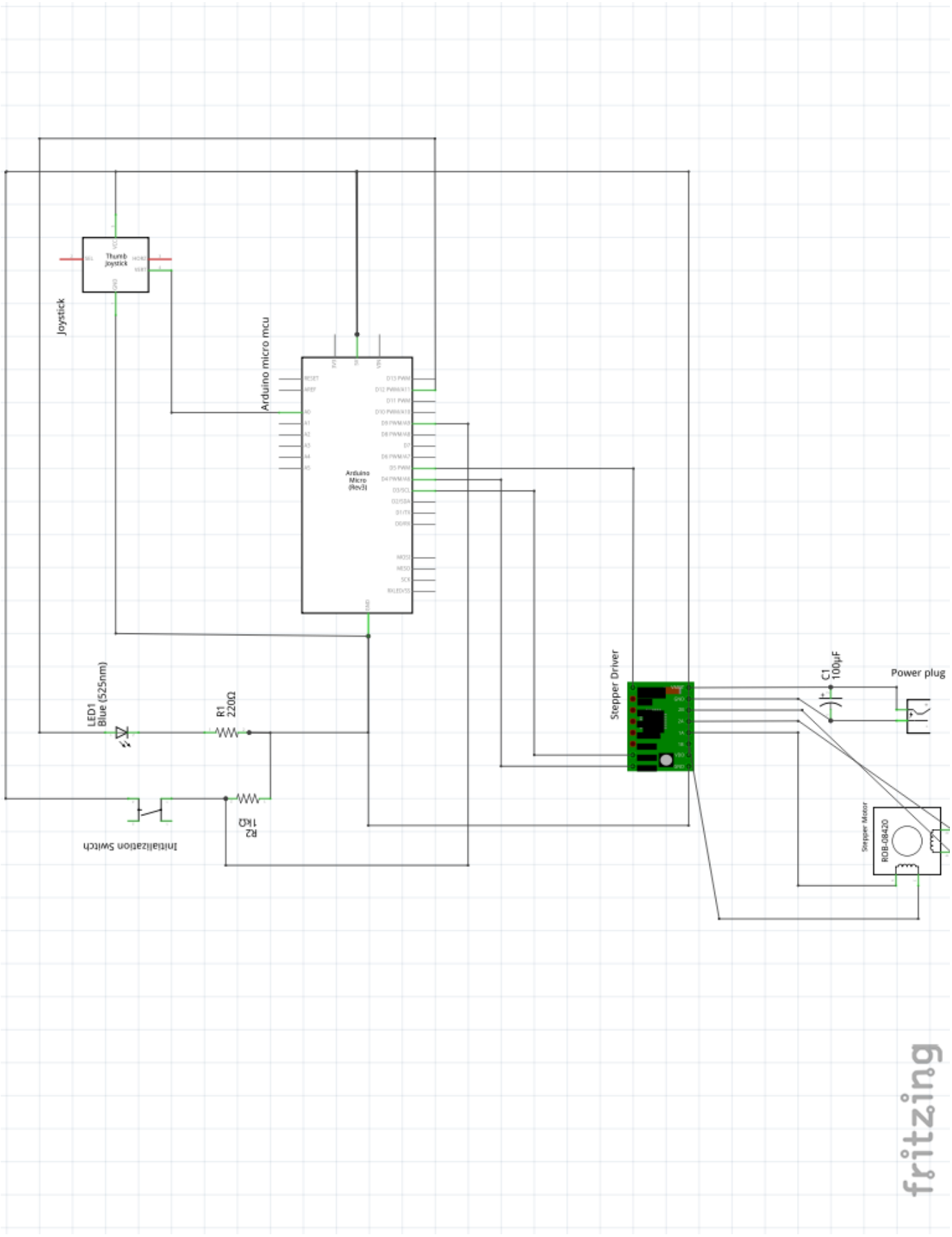




SECTION G-G

Ø65

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:				FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE				TITLE:	
DRAWN								
CHKD								
APPV'D								
MFG								
Q.A				MATERIAL:			DWG NO. vash kameras	A4
				WEIGHT:			SCALE:1:1	SHEET 1 OF 1


Joystick
Thumb Joystick
Arduino micro mcu
Arduino Micro (Rev3)
LED1 Blue (525nm)
R1 220Ω
R2 1kΩ
Initialization Switch
Stepper Driver
C1 100µF
Power plug
Stepper Motor
ROB-08420
fritzing

2. Source Code

- **Image Acquisition (*imageAcquistion.py*)**

```
import cv2
import time
from picamera.array import PiRGBArray
from picamera import PiCamera
import numpy
from subprocess import call

FPS = 10
time.sleep(30)
camera = PiCamera()
camera.led = False
capturedImg = PiRGBArray(camera)
path = "/var/www/stream/frame"    #path where the images are stored

while True:
   #grab a frame from the cam
   camera.capture(capturedImg, format="bgr")
   img = capturedImg.array
   capturedImg.truncate(0)
   #save a file in the path directory named frame.jpg and frame.bmp
   cv2.imwrite(path+".jpg",img)
   cv2.imwrite(path+".bmp",img)
   #wait for 1000/FPS miliseconds
   cv2.waitKey(1000/FPS)
```

- **Find the correct Arduino board (*Test_com.py*)**

```
import serial
import glob

filepath = 'serialPort.dat'   #name of the file where the founded serial port is written
def test_com(s):
#function that sends a the character T over serial bus and waits for the OK answer
   s.write('T')
   ans = s.read(2)
   if ans=='OK':
      return True
   else:
      return False
   s.flushInput()            #flush input serial buffer
   s.flushOutput()           #flush output serial buffer

found = False
ports = glob.glob('/dev/ttyACM*')  #list all the serial devices (starting with ACM)
for port in ports:
      try:
         s = serial.Serial(port, 9600, timeout=2)         #begin serial communication with with 'port' device. The timeout
for answer is set at 2 seconds
         if test_com(s):      #if the device responsed the correct message within the 2 seconds
                  s.close()
                  arduino_port = port          #then it is our arduino device
                  found = True
```

```
            break

        except (OSError, serial.SerialException):          #if no serial device is reached over this port
            pass                                           #pass
if found:
            print('Found arduino board on port '+str(arduino_port))
            comfile = open(filepath,'w')          #write the founded port/device on the serialPort.dat file
            comfile.write(str(port))
            comfile.close()
else:
            print('No arduino board found')
```

- **Serial Communication (*Com.py*)**

```
import argparse
import serial
import glob

parser = argparse.ArgumentParser()
parser.add_argument('-X', type=int, nargs='?')
parser.add_argument('-Y', type=int, nargs='?')
parser.add_argument('-Z', type=int, nargs='?')
parser.add_argument('-A', type=int, nargs='?')
args=parser.parse_args()

try:
    filepath = 'serialPort.dat'
    comfile = open(filepath,'r')
    port = comfile.read()                #read the serial port from the serialPort.dat file
    comfile.close()

    arg_list = vars(args)                #read the arguments list
    sr = serial.Serial(port,9600)        #begin serial communication
    for name in arg_list:
        if (arg_list[name]!=None):
            print('Steps to perform = ' + str(arg_list[name]))
            sr.write(name+str(arg_list[name]))   #echo the argument to the serial port
except(OSError, serial.SerialException):
    port = port.strip()          #no serial device error
    print('Cannot find serial device '+port+'! Please run again find device script!')
except getattr(__builtins__,'FileNotFoundError', IOError):     #file not found error for serialPort.dat file
    print('Cannot find serialPort.dat file. Please run again find device script!')
```

- **Auto Focus (*autofocus.py*)**

```
import cv2
import serial
import time
import argparse

N_STEPS = 200 #value of steps to be performed between each discrete position

parser = argparse.ArgumentParser()
parser.add_argument('-A', type=int, nargs='?')
args=parser.parse_args()
```

```
pr_bv = 0          #varible to store the laplacian varible of the previous position

try:
   filepath = 'serialPort.dat'          #filename of the file where the correct serial port is written
   comfile = open(filepath,'r')
   port = comfile.read()                #read the serial port from the serialPort.dat file
   comfile.close()
   sr = serial.Serial(port,9600)        #construct a Serial oblect
   imagePath='../stream/frame'          #path of the image to be processed

   command='A0'

   arg_list = vars(args)      #read the argument list
   if(arg_list['A']!=None):
          command = 'A'+str(arg_list['A'])

   sr.write(command)        #give the appropriate value to Arduino next to A argument in slecting initialization from Z=0
or current Z
   i = 0
   while (True):
      strg = sr.read(4)        #read 4 bytes frim the serial buffer ('OK' response from Arduino)
      sr.flushInput()
      sr.flushOutput()
      imagePath1 = imagePath+str(i)+'.jpg'   #change this line of code if the current frame is needed to be processed with
imagePath1 = imagePath+'.bmp'
      time.sleep(1)
      image = cv2.imread(imagePath1)
      gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) #transform the image to grayscale
      bv = cv2.Laplacian(gray, cv2.CV_64F).var()              #compute the variance of the laplacian of this image
      if (pr_bv >= bv):                                       #if the previous image is less blurred
         sr.write('Z-'+str(N_STEPS))                          #tell the Arduino to return the stage to the previous
position
         i = i-1
         break                                          #break the loop
      pr_bv = bv
      sr.write('Z'+str(N_STEPS))
      i = i+1                                                 #comment this line of code if the current frame is needed
to be processed
   sr.write('A0')                                             #when the loop ends turn the light off
except(OSError, serial.SerialException):     #no serial device error
   port = port.strip()
   print('Cannot find serial device '+port+'! Please run again find device script!')
except getattr(__builtins__,'FileNotFoundError', IOError):     #file not found error for serialPosrt.dat file
   print('Cannot find serialPort.dat file. Please run again find device script!')
```

- **Web Interface (*index.php*)**

```
<html>
<head>
<title>Web Interface</title>
</head>

<body>
<h1>Web interface to remotely control the microscope</h1>

<p align=center>
```

```
<strong>Info:  </strong>
<?php
$Zstep = 0;

function onZstepAdjust() {
  global $Zstep;   //number of steps to perform
  $sel = $_POST[ZstepList];          //the selected value from the list
  switch($sel) {
  case "st1":
    $Zstep = 1;
    break;
  case "st2":
    $Zstep = 5;
    break;
  case "st3":
    $Zstep = 10;
    break;
  case "st4":
    $Zstep = 50;
    break;
  case "st5":
    $Zstep = 100;
    break;
  case "st6":
    $Zstep = 200;
    break;
  case "st7":
    $Zstep = 400;
    break;
  default:
    $Zstep = 1;
  }
}
if(isset($_POST[AddX])) {
  onZstepAdjust();
  echo exec('python Com.py -Z'.(string)$Zstep);        //run the Com python script to transmit the command to Arduino
}
if(isset($_POST[RemoveX])) {
  onZstepAdjust();
  echo exec('python Com.py -Z'.(string)(-$Zstep));     //run the Com.py to transmit the command to Arduino
}
if(isset($_POST[SaveImage])) {
  $flnm = date('d-m-y_H:i:s');         //name of the file according to current date and time
  echo exec('cp -v ../stream/frame.bmp '.$flnm.'.bmp');            //copy the current frame.bmp file to the working directory
giving the filename defined above
}
if(isset($_POST[DownloadZip])) {
  echo exec('zip -FS Images.zip *.bmp');      //create a zip file conataining all the bmp files in current directory replacing
any older ones
  header('Content-type: application/zip');
  header('Content-Disposition: attachment; filename="Images.zip"');
  readfile('Images.zip');
}
if(isset($_POST[findDevice])) {
  echo exec('python testCom.py');   //run testCom python script to find on which serial port is connected the Arduino
}
if(isset($_POST[Auto_focus])) {
  if(isset($_POST[Auto_start])) {
```

```
    $val = $_POST[Auto_start];
    if ($val=="Z0")
      echo exec('python autofocus.py -A1');               //run autofocus script. Argument 1 means start from Z=0 position
    else if ($val=="Here")
      echo exec('python autofocus.py -A-1');  //run autofocus script. Argument -1 means start from current position
   }
   else
    echo 'Please select starting point first!';
}

if(isset($_POST[ClearStore])) {
  exec('rm *bmp');                                        //clear any bmp files in the working directory
  echo('Images cleared');
}
// HTML code is following
?>

<br>
</p>
<p align=center>
<img src="/Rpi?action=stream" name="video"></img>
</p>
<form action="<?php echo($_SERVER[PHP_SELF]) ?>" method="POST">
<p align=center><strong>Z command:</strong><br>
<input type="Submit" name="AddX",  value="Up">
<input type="Submit" name="RemoveX",  value="Down">
<select name="ZstepList">
  <option value="st1" <?php if($_POST['ZstepList']=="st1") echo 'selected'?>>1 step (0.9 deg)</option>
  <option value= "st2"<?php if($_POST['ZstepList']=="st2") echo 'selected'?>>5 steps (4.5 deg)</option>
  <option value="st3"<?php if($_POST['ZstepList']=="st3") echo 'selected'?>>10 steps (9 deg)</option>
  <option value="st4"<?php if($_POST['ZstepList']=="st4") echo 'selected'?>>50 steps (45 deg)</option>
  <option value="st5"<?php if($_POST['ZstepList']=="st5") echo 'selected'?>>100 steps (90 deg)</option>
  <option value="st6"<?php if($_POST['ZstepList']=="st6") echo 'selected'?>>200 steps (180 deg)</option>
  <option value="st7" <?php if($_POST['ZstepList']=="st7") echo 'selected'?>>400 steps (360 deg)</option>
</select>
<br>
</p>
<p align = center>
<strong>Click to find serial device:</strong><br>
<input type="Submit" name = "findDevice" value="Find serial device">
</p>
<p align = center>
<strong>Images Storing:</strong><br>
<input type="Submit" name="SaveImage" value="Save Image on RPi's Disk">
<input type="Submit" name="DownloadZip" value="Zip and download saved images">
<input type="Submit" name="ClearStore" value="Clear saved images">
</p>

<p align=center><strong>Auto Focus Mode</strong><br>
<input type="Submit" name="Auto_focus" value="Auto Focus"><br>
Start from Z=0:<input type = "radio" name="Auto_start" value="Z0" <?php if($_POST['Auto_start']=="Z0") echo
'checked'?>><br>
Start from here:<input type = "radio" name="Auto_start" value="Here" <?php if($_POST['Auto_start']=="Here") echo
'checked'?>><br>
</p>
</form>
</body>
```

- **Arduino Sketch (*Final_Sketch.ino*)**

```
#include <AccelStepper.h>
#include <MultiStepper.h>

void Zstep(int newPos);
void joystickWatcher();
bool checkAllowedDistance(int newPos);

//pin assignments
int ledPin = 12;
int switchPin = 9;

int stepPin = 3;
int dirPin = 4;
int enPin = 5;

char auto_mode = -1;
int led_state = LOW;
int switchState = 0;

int zJoystick = A0;
int steps = 0;

const int RPM = 20; //motor's speed in RPM
const int maxAllowedPos = 2400; //constant to define max allowed steps

AccelStepper Zstepper(1, stepPin, dirPin);

void setup() {

  Serial.begin(9600);

  pinMode(ledPin,OUTPUT);
  pinMode(enPin,OUTPUT);
  pinMode(switchPin, INPUT);

  digitalWrite(ledPin,led_state);
  digitalWrite(enPin,LOW);
  Zstepper.setMaxSpeed(1500);

  //initialization of the stage position
  while(switchState == 0) {  //until the switch is pressed
    switchState = digitalRead(switchPin);
    Zstepper.setSpeed(-RPM);  //move downwards the stage
    Zstepper.runSpeed();
  }
  Zstepper.setCurrentPosition(0);  //current position is Z=0

}


void loop() {
  int newPos;
  //Serial.println(Zstepper.currentPosition());

  joystickWatcher();  //watch for joystick command
  steps = 0;
```

```
  if (Serial.available() > 0) {  //if bytes are available to serial bus
    char start_char = Serial.read();  //read the first byte-character
    switch(start_char) {
      case('Z'):
        steps = Serial.parseInt();  //integer that represents number of steps
        newPos = Zstepper.currentPosition()+steps;  //define the new position where we want the stage to be
        Zstep(newPos);  //step until this new position
        break;
      case('A'):
        auto_mode = Serial.parseInt();  //integer that represetns the position to start the autofocus procedure. If it's zero the
LED is turned off
        if (auto_mode==1) {  //if we are in autofocus mode starting from Z=0
          led_state = HIGH;  //turn on the led
          Zstep(0);          //step to Z=0
          delay(30);
          Serial.println("OK");
        }
        else if (auto_mode==-1) {
          led_state=HIGH;  //if we are in autofocus mode starting from current Z
          delay(30);
          Serial.println("OK");

        }
        else
          led_state=LOW;  //turn off the LED
        break;
      case('T'):                //the chracter T represents the serial communication testing
        Serial.println("OK");   //OK response required
    }
  }

  digitalWrite(ledPin,led_state);
  delay(1);
}

void joystickWatcher() {
 //this function reads the analog input whre the joysticj is connected and maps the read value to interval [-5,5] that
represents 5 different speeds on each direction
  int joy = analogRead(zJoystick);
  joy = map(joy,0,1023,-5,5);

  if (joy!=0) {
    Zstepper.setSpeed(10*joy);
    //Serial.println(10*joy);
    if (checkAllowedDistance(Zstepper.currentPosition()+joy/abs(joy)))  //if we have not overpassed the maximum Z
position
      Zstepper.runSpeed();
    int dt = 70-10*abs(joy);
    delay(dt);  //by changing the delay time the speed of the motor is adjusted
  }
}


void Zstep(int newPos) {

  if(!checkAllowedDistance(newPos))  //if the new position is going to be higher than the maximum allowed no action is
taken
    return ;
```

```
  Zstepper.moveTo(newPos);  //set new position to move
  Zstepper.setSpeed(RPM);    //set the speed to RPM rpm

  while(Zstepper.distanceToGo() !=0) {
    Zstepper.runSpeedToPosition();  //run the stepper with the predifined speed until the willing position is reached
    switchState = digitalRead(switchPin);
    if (switchState) break;
  }
  if (auto_mode!=0)  {
    Serial.println("OK");
    delay(30);
}
}

bool checkAllowedDistance(int newPos) {
  //this function checks if we have overpassed the maximum allowed Z position and returns false if this occurs, else
returns true
  bool check = newPos < maxAllowedPos;
  if (!check) {  //if we have overpassed the max postition blink the indicating led
    digitalWrite(ledPin,!led_state);
    delay(50);
    digitalWrite(ledPin, led_state);
  }
  return check;
}
```